MITSUBISHI ELECTRIC RESEARCH LABORATORIES
Cambridge, Massachusetts

Divide & Concur and Difference-Map Belief Propagation

Jonathan Yedidia

Yige Wang
Stark Draper

Outline

- Review of factor graphs for optimization and inference, and the min-sum Belief Propagation (BP) algorithm

- Gravel and Elser's "Divide & Concur" algorithm interpreted as a message-passing algorithm

- Decoders for Low-Density Parity Check (LDPC) Codes
  - Divide & Concur Decoder
  - "Difference-Map Belief Propagation" (DMBP) Decoder

- Simulation Results
  - *DMBP Decoder significantly improves error-floor performance compared with standard BP decoders, with similar complexity!*

Probabilistic Inference and Optimization Problems

- ***Channel Coding***: Data is corrupted by a noisy channel. What is the most probable version of the original data?
- ***Computer Vision***: A camera captures an ambiguous scene. What is the most probable interpretation of the scene?
- ***Physics***: An atomic-scale energy function is given for a molecule or crystal. What is the most probable configuration?
- ***Optimization***: We are given a problem with constraints and costs. What is the lowest cost configuration consistent with the constraints?

- Equivalence of probabilistic inference and optimization problems:

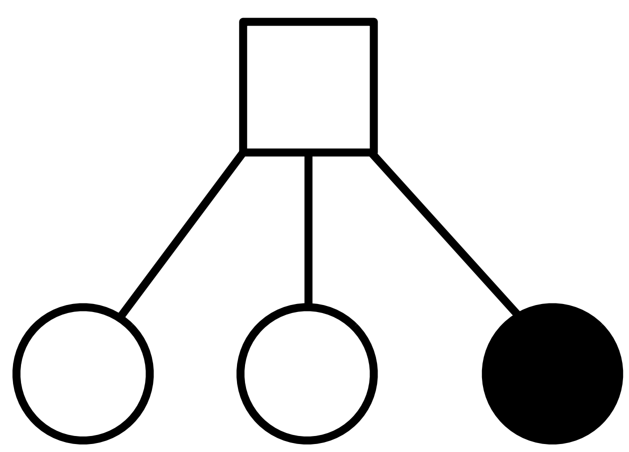
$$probability(X) \propto e^{-cost(X)}$$

Representing Costs (or Probabilities) in a Factor Graph

- We assume that the overall cost is the sum of M local costs

$$Cost = \sum_{a=1}^{M} C_a(X_a)$$

- We represent local cost functions with squares (called "factor nodes"), connected to the circles representing variable nodes involved in the local cost function.



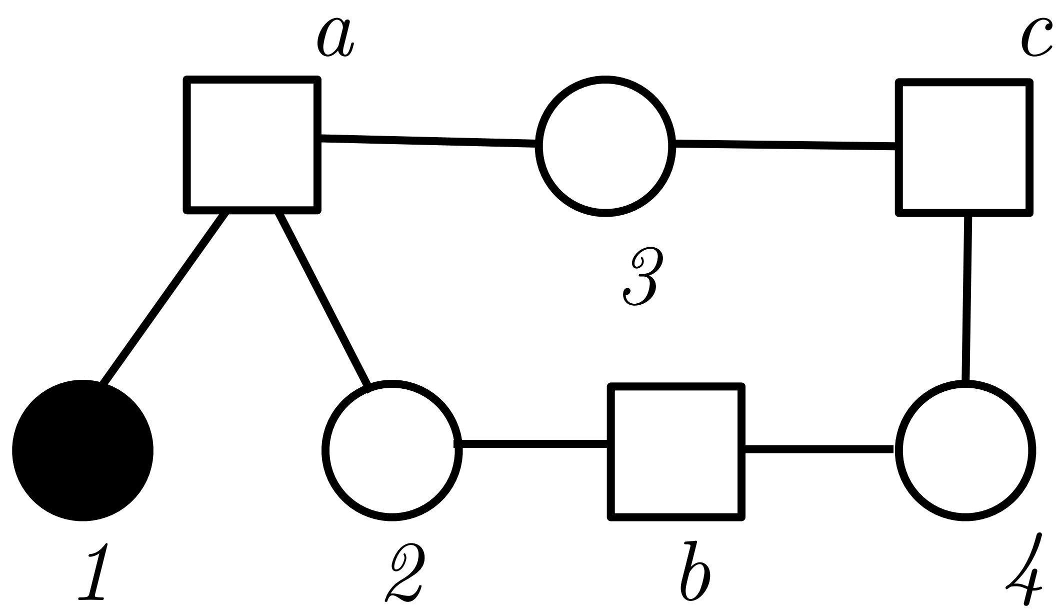
hidden variable

observed variable

Example

$$Cost = C_a(x_1, x_2, x_3) + C_b(x_2, x_4) + C_c(x_3, x_4)$$

x_1	x_2	x_3	C_a
0	0	0	∞
0	0	1	0
0	1	0	0
0	1	1	∞
1	0	0	0
1	0	1	∞
1	1	0	∞
1	1	1	0



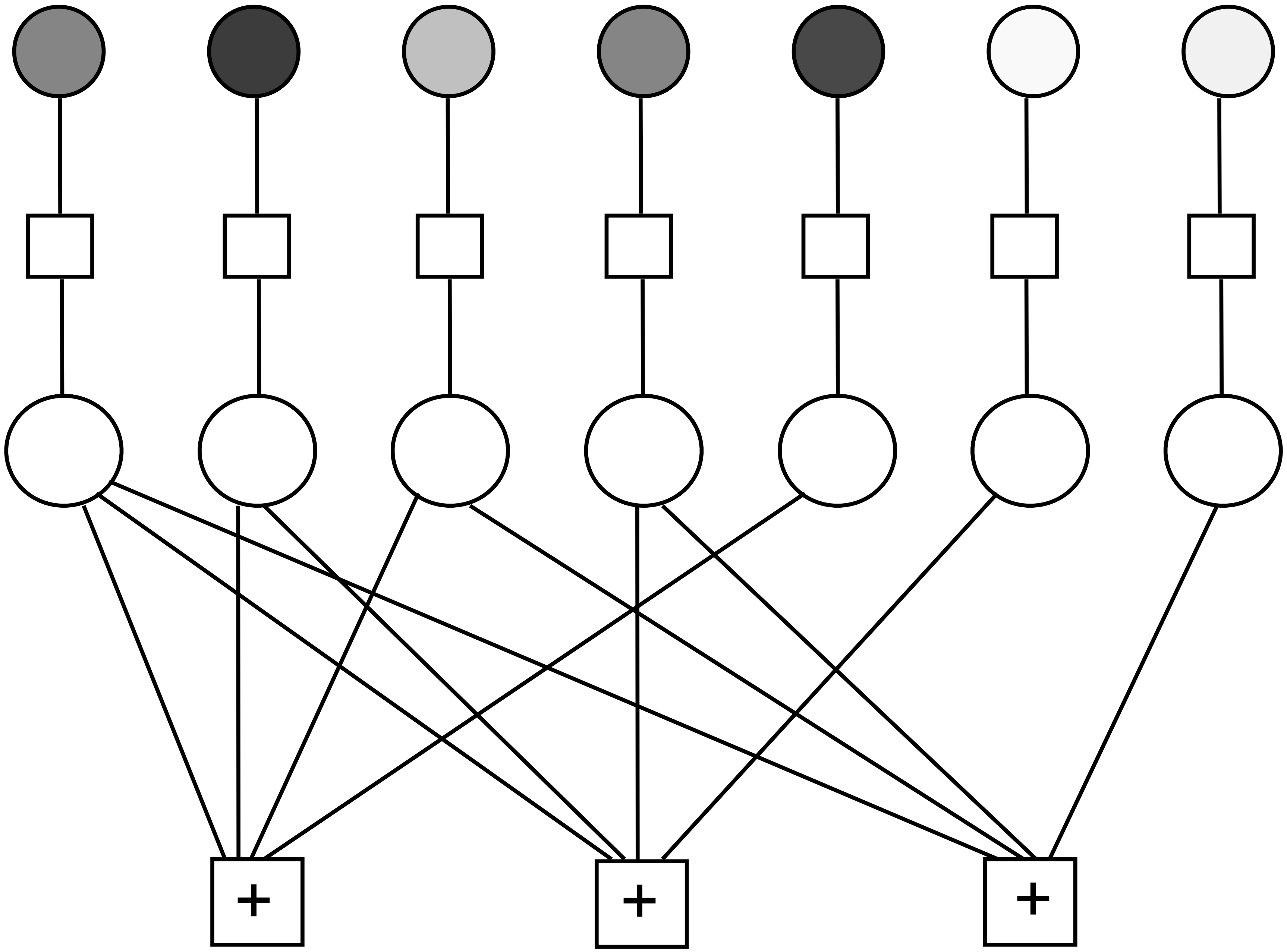
x_3	x_4	C_c
0	0	0.4
0	1	1.9
0	2	0.2
1	0	4.9
1	1	0.3
1	2	2.4

x_2	x_4	C_b
0	0	1.2
0	1	1.7
0	2	3.2
1	0	1.9
1	1	0.6
1	2	1.4

Infinite cost configurations are *forbidden* in *"hard"* constraints.

Error-correcting Codes

(Tanner, 1981)



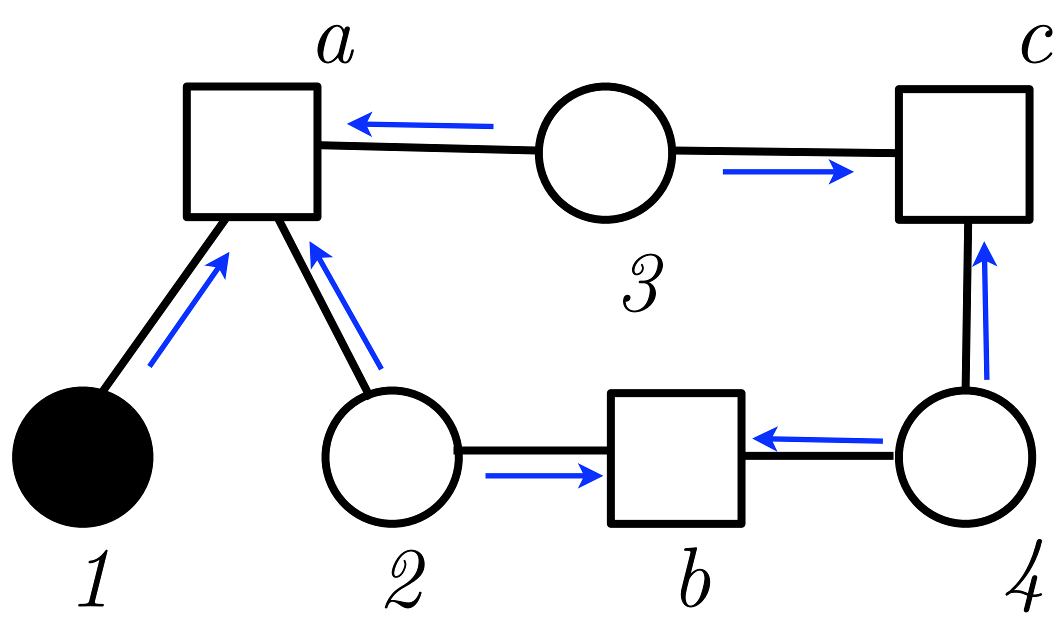
Observed Symbols

Unknown Transmitted Bits

Parity Checks

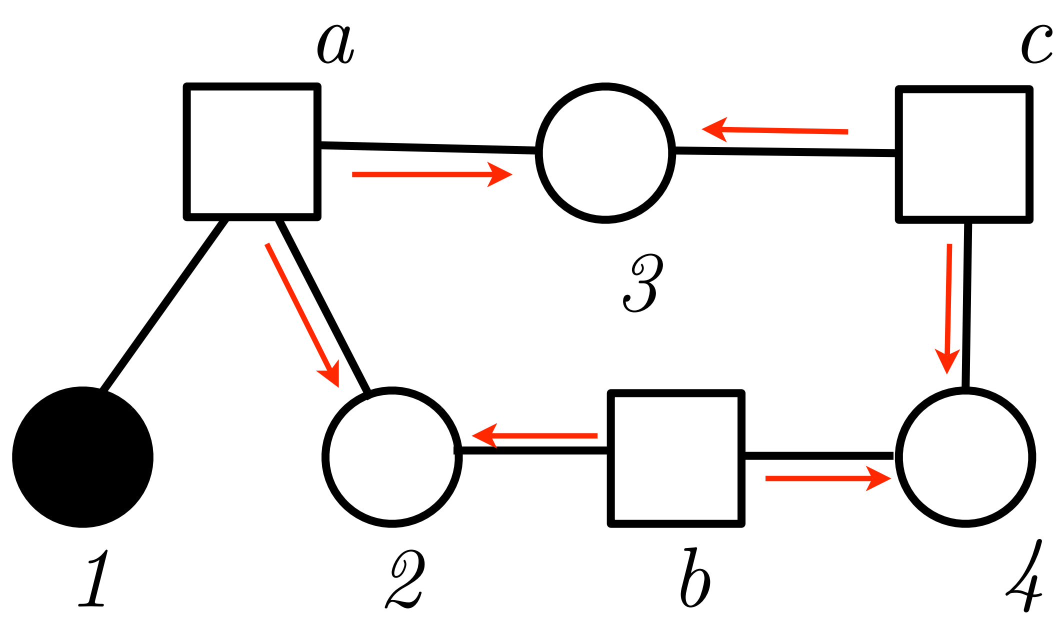
Goal: find most probable code-word

Overall Structure of Message-Passing Algorithms



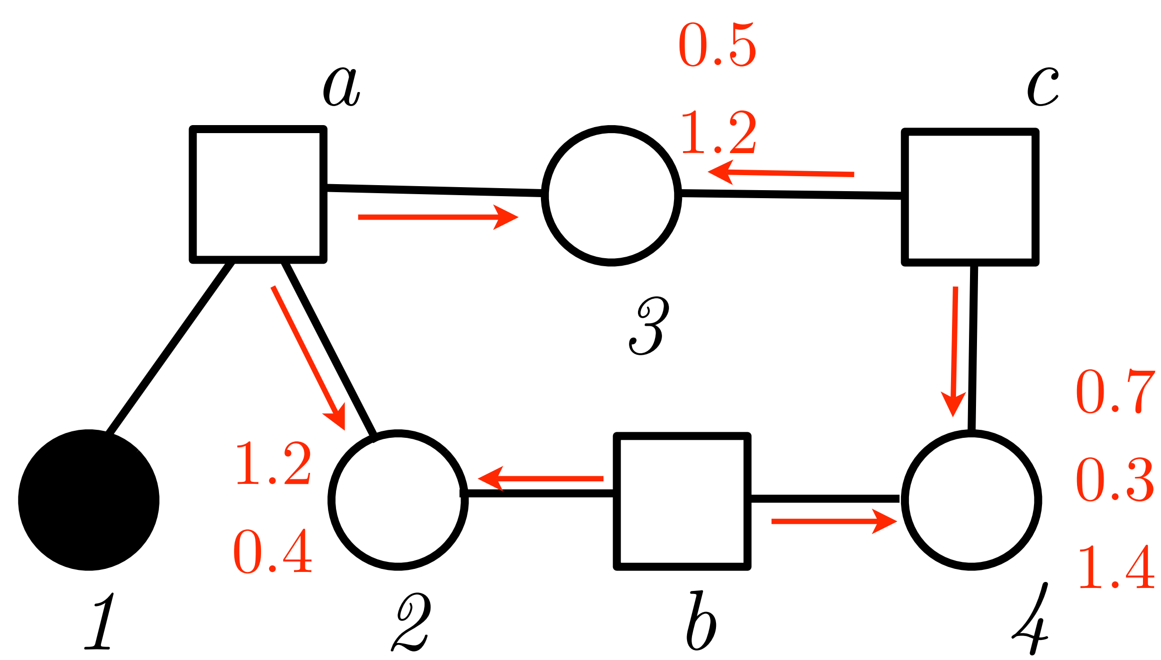
1. Initialize messages from variable nodes to factor nodes.
2. Update messages from factors to variables.
3. Update beliefs.
4. Threshold beliefs, and check for termination.
5. Update messages from variables to factors, and go to step 2.

Overall Structure of Message-Passing Algorithms



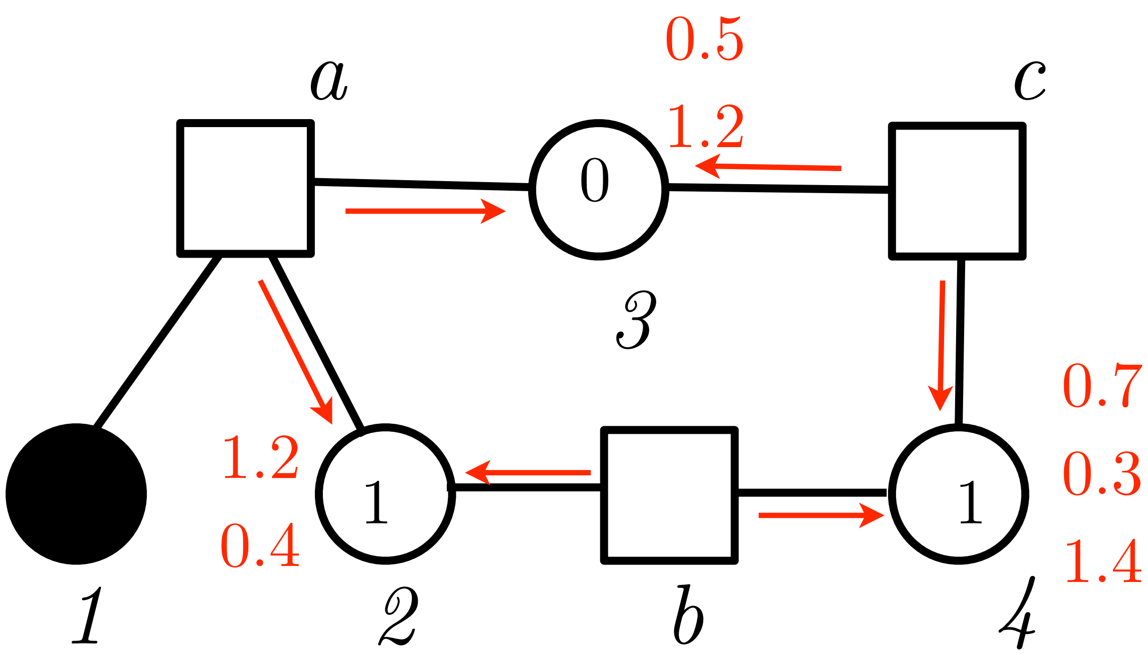
1. Initialize messages from variable nodes to factor nodes.
2. Update messages from factors to variables.
3. Update beliefs.
4. Threshold beliefs, and check for termination.
5. Update messages from variables to factors, and go to step 2.

Overall Structure of Message-Passing Algorithms



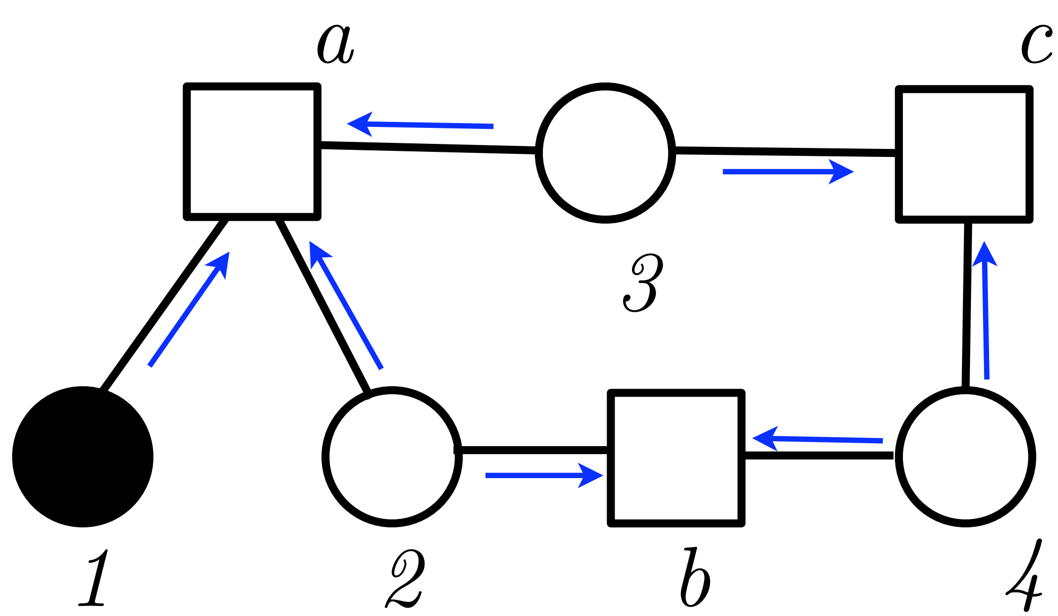
1. Initialize messages from variables to factor nodes.
2. Update messages from factors to variables.
3. Update beliefs.
4. Threshold beliefs, and check for termination.
5. Update messages from variables to factors, and go to step 2.

Overall Structure of Message-Passing Algorithms



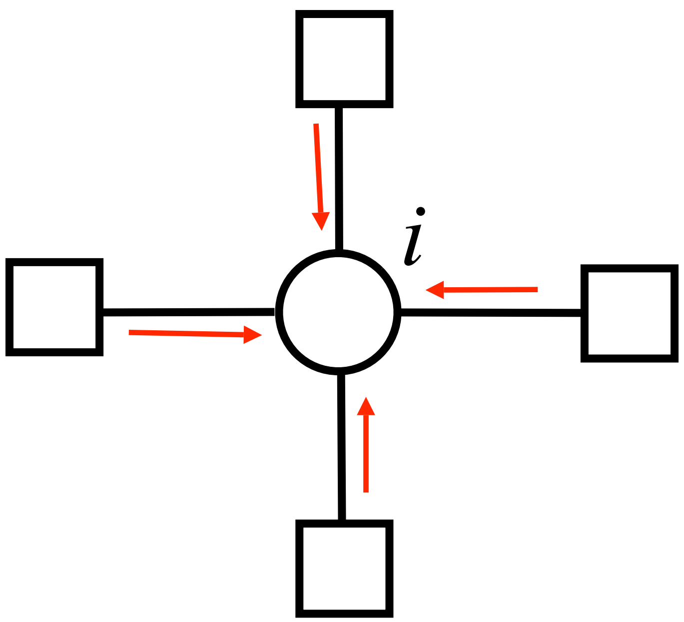
1. Initialize messages from variable nodes to factor nodes.
2. Update messages from factors to variables.
3. Update beliefs.
4. Threshold beliefs, and check for termination.
5. Update messages from variables to factors, and go to step 2.

Overall Structure of Message-Passing Algorithms



1. Initialize messages from variable nodes to factor nodes.
2. Update messages from factors to variables.
3. Update beliefs.
4. Threshold beliefs, and check for termination.
5. Update messages from variables to factors, and go to step 2.

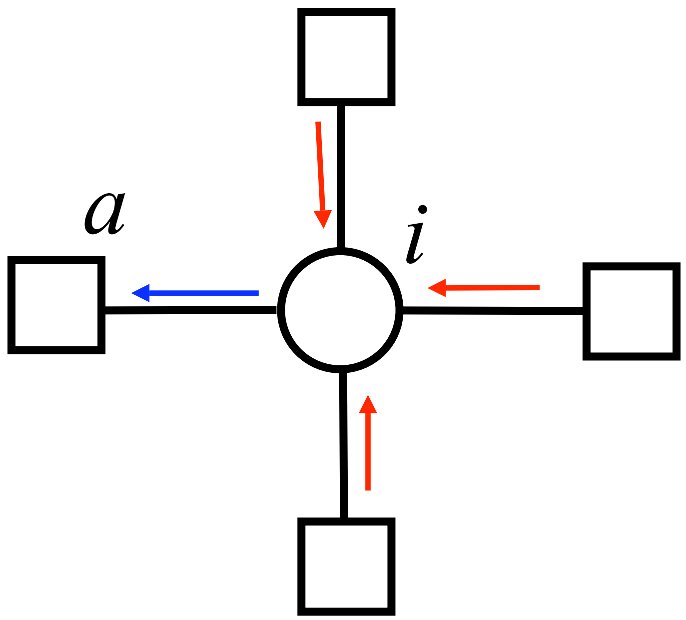
Belief Propagation Belief Update Rules



$$b_i(x_i) = \sum_{a \in N(i)} m_{a \rightarrow i}(x_i)$$

"belief" "messages"

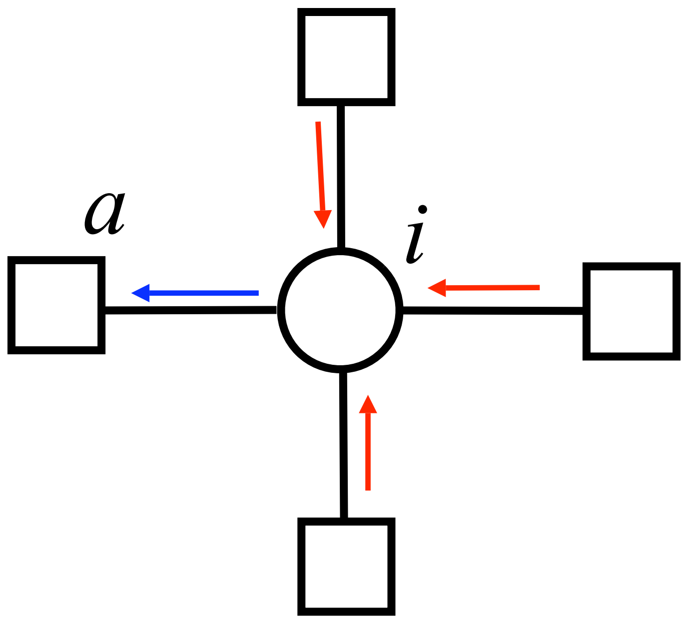
Belief Propagation Message Update Rules



$$m_{i \to a}(x_i) = \sum_{b \in N(i) \setminus a} m_{b \to i}(x_i)$$

A variable node tells nearby factor nodes what it thinks its costs will be for being in different states.

Belief Propagation Message Update Rules

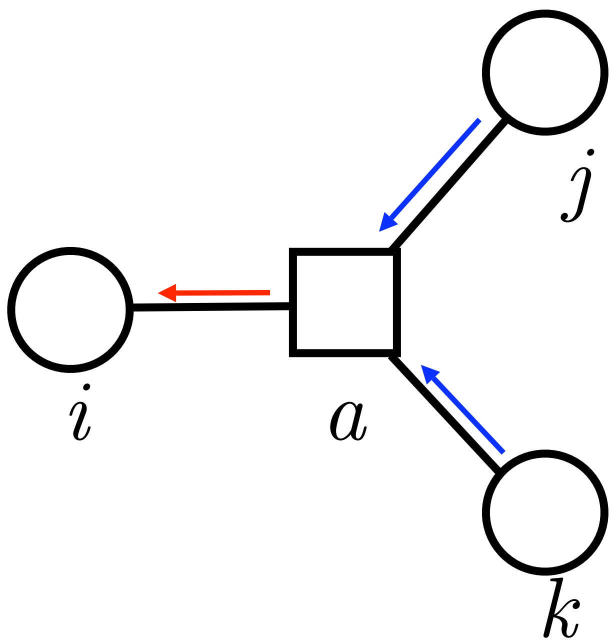


$$m_{i \to a}(x_i) = \sum_{b \in N(i) \backslash a} m_{b \to i}(x_i)$$

$$m_{i \to a}(x_i) = b_i(x_i) - m_{a \to i}(x_i)$$

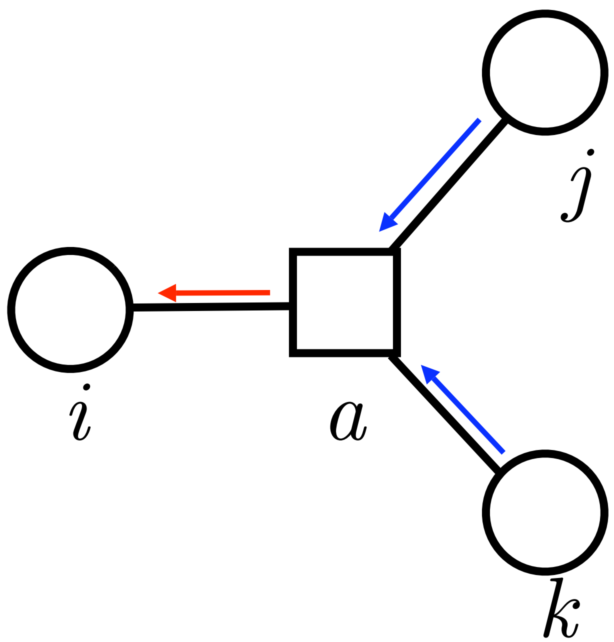
A variable node tells nearby factor nodes what it thinks its costs will be for being in different states.

Belief Propagation Message Update Rules



$$m_{a\to i}(x_i) = \min_{x_j, x_k} \left[C_a(x_i, x_j, x_k) + m_{j\to a}(x_j) + m_{k\to a}(x_k) \right]$$

Belief Propagation Message Update Rules



$$m_{a\to i}(x_i) = \min_{X_a \setminus x_i} \left[C_a(X_a) + \sum_{j \in N(a)\setminus i} m_{j\to a}(x_j) \right]$$

"Min-Sum Rule"

$$m_{a\to i}(x_i) = \min_{x_j, x_k} [C_a(x_i, x_j, x_k) + m_{j\to a}(x_j) + m_{k\to a}(x_k)]$$

Outline

- Review of factor graphs for optimization and inference, and the min-sum Belief Propagation (BP) algorithm
- Gravel and Elser's "Divide & Concur" algorithm interpreted as a message-passing algorithm
- Decoders for Low-Density Parity Check (LDPC) Codes
  - Divide & Concur Decoder
  - "Difference-Map Belief Propagation" (DMBP) Decoder
- Simulation Results
  - *DMBP Decoder significantly improves error-floor performance compared with standard BP decoders, with similar complexity!*

Divide & Concur

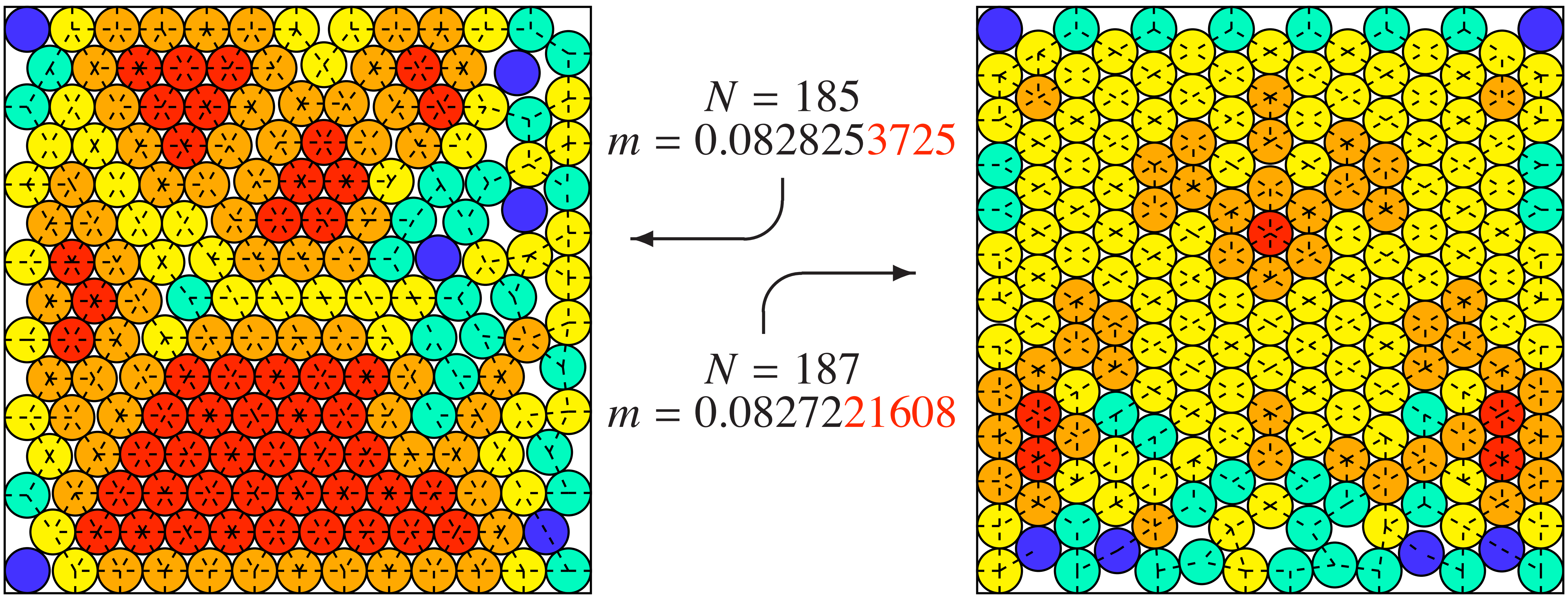
- Introduced by Simon Gravel and Veit Elser from Cornell, who generalized an approach used by X-ray crystallographers.

- In contrast with BP, works well with ***continuous-valued*** variables.

- Also works well when there is ***no local evidence*** for the variables, just constraints between the variables.

- Note: Gravel and Elser did not describe D&C as a message-passing algorithm, but it can be formulated in that way.

Sphere-packing Problem Shows Advantages Compared with Belief Propagation



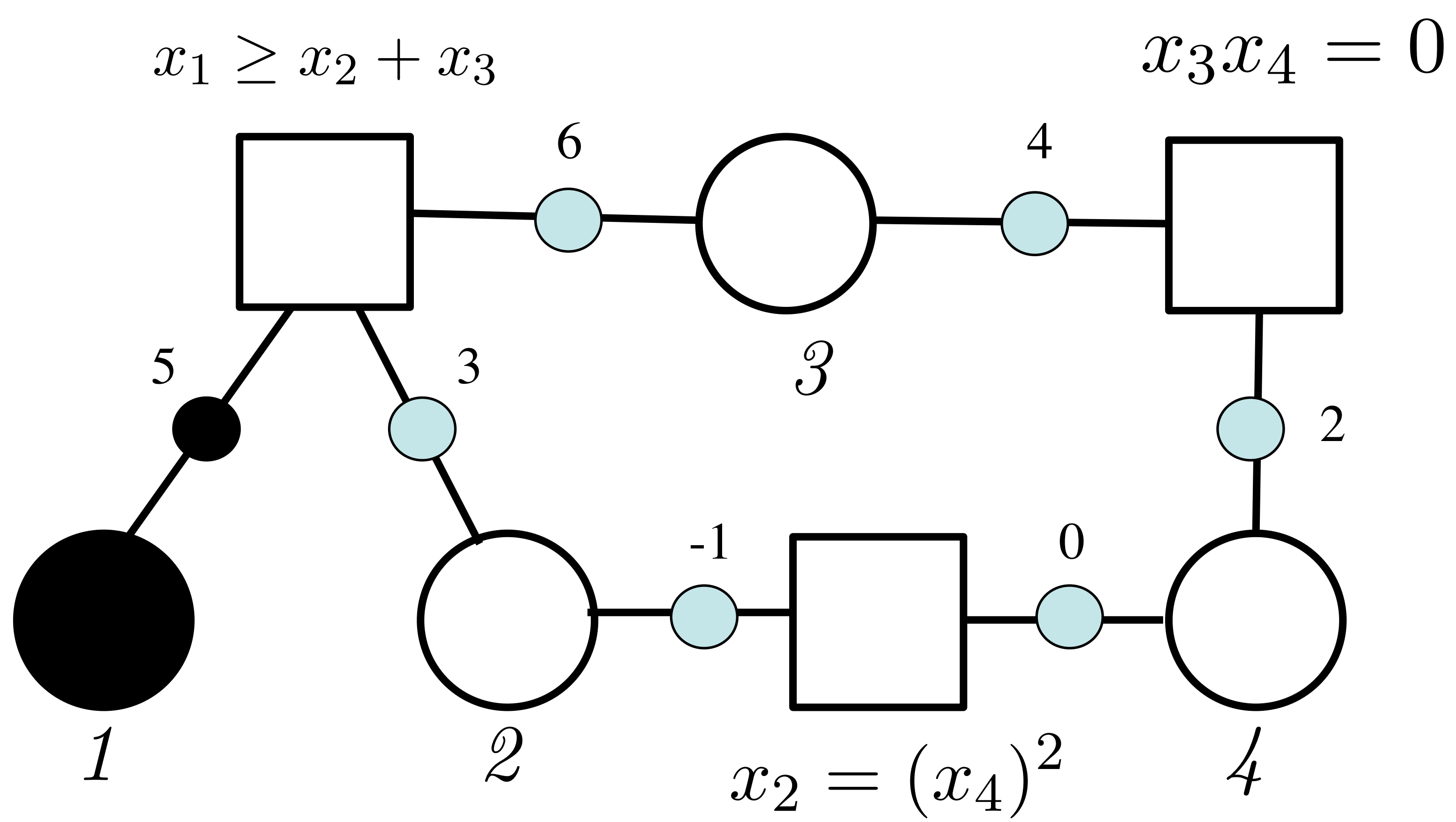
Improved packings, from Gravel's Ph.D. thesis (2009)

Continuous variables, and no local evidence

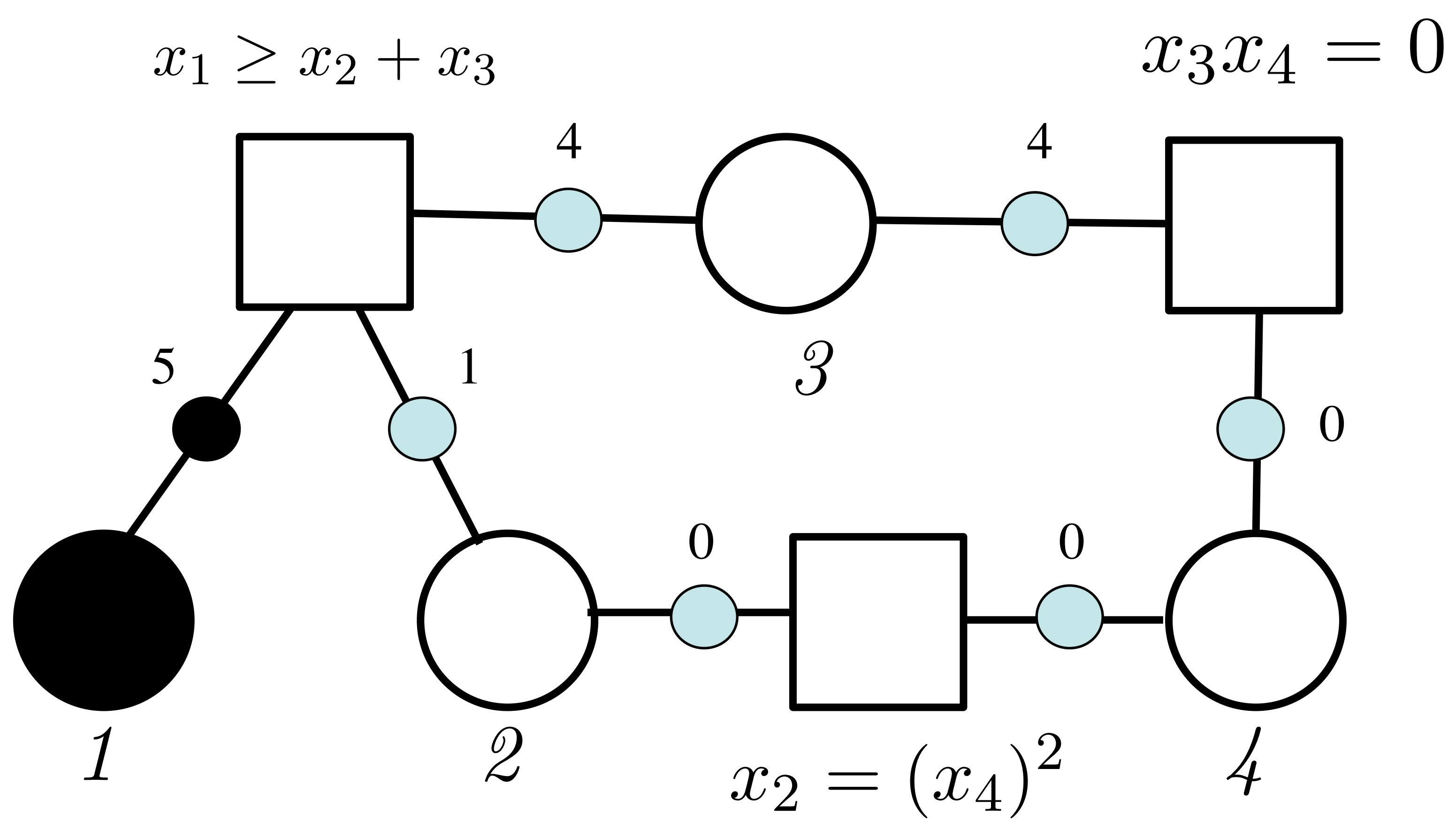
Divide & Concur Ideas

- Use only "hard" constraints on the variables. I.e., all costs at factor nodes are zero or infinity. (Problems with soft constraints can still be handled by introducing explicit "cost" variables.)
- Each variable has a "replica" for each constraint it is involved in.
- We search for a set of replica values that satisfy all the constraints ("Divide projection"), and such that all replicas for the same variable have the same value ("Concur projection").
- The Divide projection moves the replica values to the nearest values that satisfy the constraint. The Concur projection averages the replica values belonging to the same variable.
- Use Elser's "Difference-Map" dynamics to avoid local traps that occur when one naively alternately iterates between Divide and Concur projections.

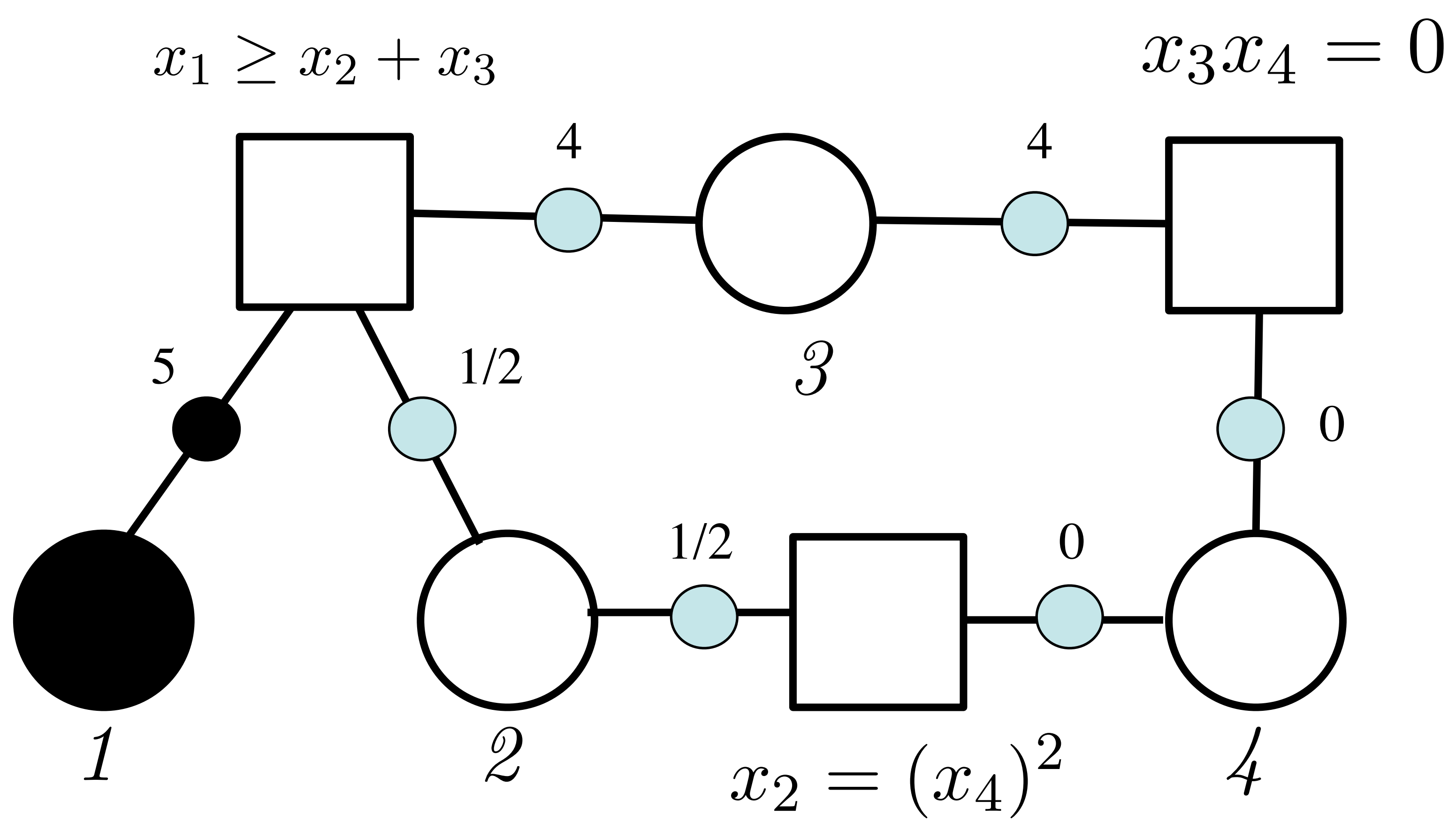
D&C Projections: Divide Projection



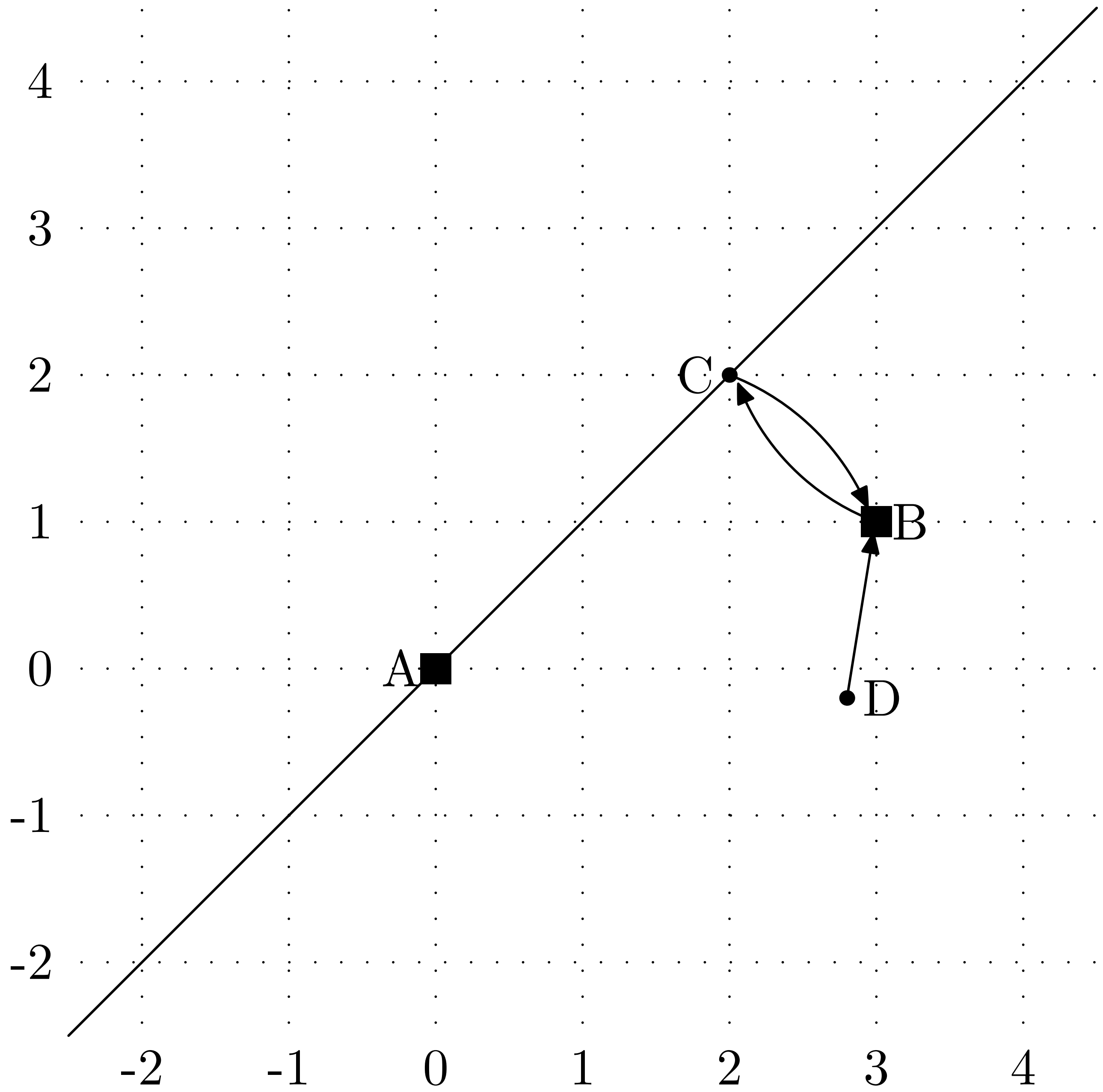
D&C Projections: Concur Projection



D&C Projections

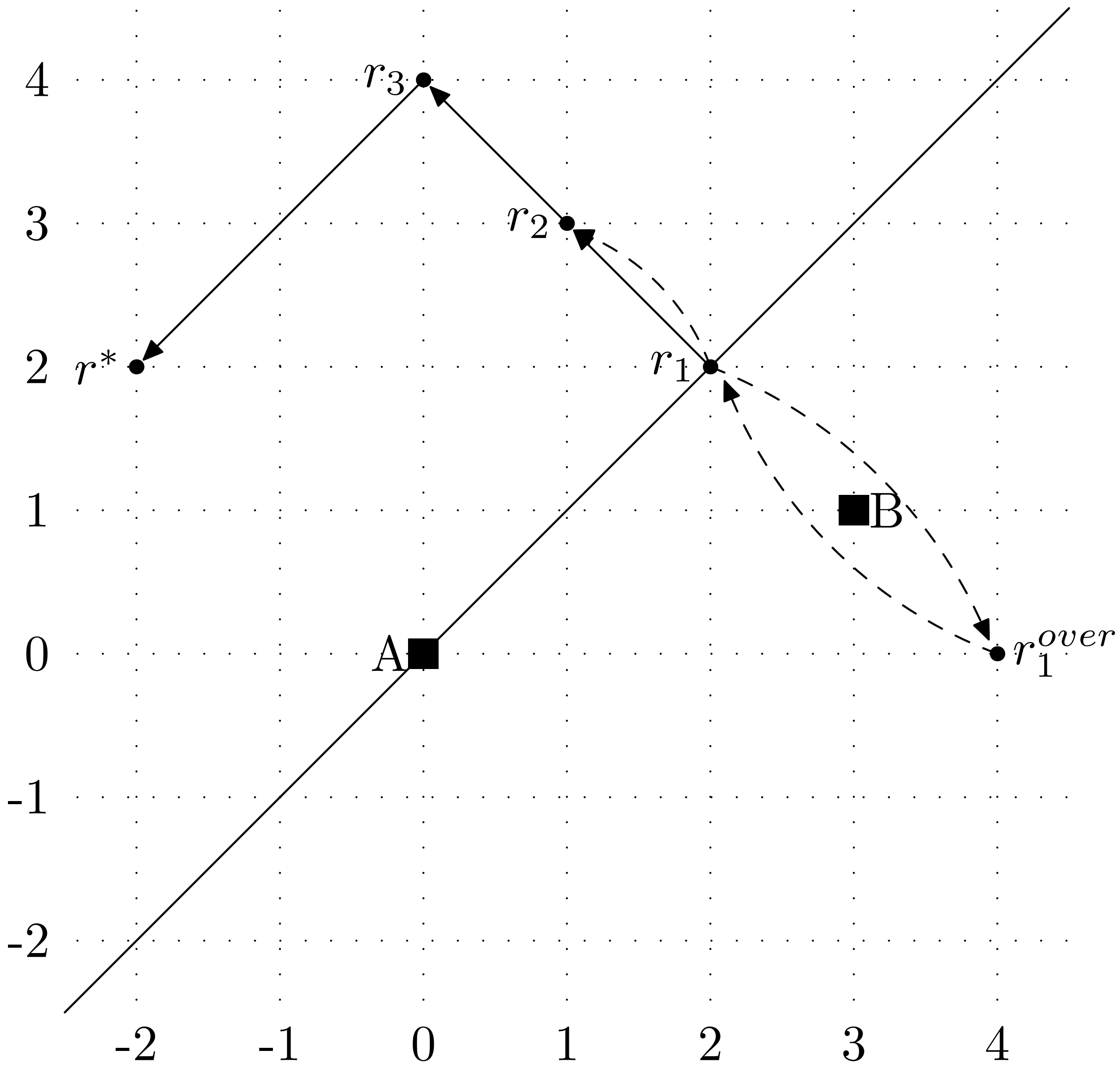


Traps in Naive Alternating Projection Approach



$$\boldsymbol{r}_{t+1} = P_C(P_D(\boldsymbol{r}_t))$$

Difference-Map Dynamics



$$\boldsymbol{r}_{t+1} = P_C\left(\boldsymbol{r}_t + 2[P_D(\boldsymbol{r}_t) - \boldsymbol{r}_t]\right) - [P_D(\boldsymbol{r}_t) - \boldsymbol{r}_t]$$

t	$\boldsymbol{r}_t$	$P_D(\boldsymbol{r}_t)$	$\boldsymbol{r}_t^{over}$	$\boldsymbol{r}_t^{conc}$
1	$(2,2)$	$(3,1)$	$(4,0)$	$(2,2)$
2	$(1,3)$	$(3,1)$	$(5,-1)$	$(2,2)$
3	$(0,4)$	$(0,0)$	$(0,-4)$	$(-2,-2)$
4	$(-2,2)$	$(0,0)$	$(2,-2)$	$(0,0)$
5	$(-2,2)$			

Difference-Map Dynamics:

- Overshoot
- Concur
- Correct

Divide & Concur As Message-Passing

- "Overshoot" replica values are messages from constraints to variables.

$$\boldsymbol{m}_{a\rightarrow}(t) = \boldsymbol{m}_{\rightarrow a}(t) + 2[P_D^a(\boldsymbol{m}_{\rightarrow a}(t)) - \boldsymbol{m}_{\rightarrow a}(t)]$$

- "Concurred" replica values are beliefs.

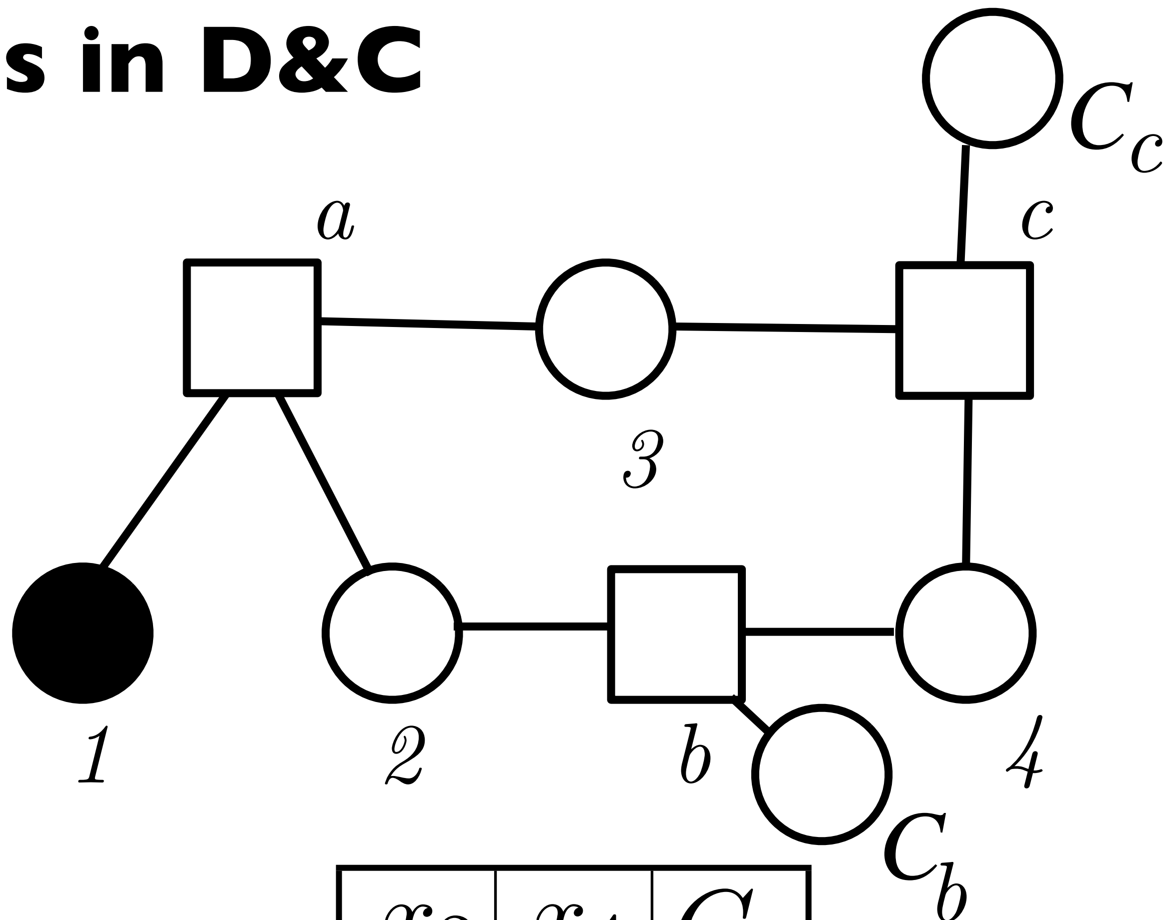
$$b_i(t) = P_C^i(\boldsymbol{m}_{\rightarrow i}(t)) = \frac{1}{|\mathcal{M}(i)|}\sum_{a\in\mathcal{M}(i)} m_{a\rightarrow i}(t)$$

- "Corrected" replica values are messages from variables to checks.

$$m_{i\rightarrow a}(t+1) = b_i(t) - 1/2\,[m_{a\rightarrow i}(t) - m_{i\rightarrow a}(t)]$$

Soft Constraints in D&C

x_1	x_2	x_3	C_a
0	0	0	∞
0	0	1	0
0	1	0	0
0	1	1	∞
1	0	0	0
1	0	1	∞
1	1	0	∞
1	1	1	0



x_3	x_4	C_c
0	0	0.4
0	1	1.9
0	2	0.2
1	0	4.9
1	1	0.3
1	2	2.4

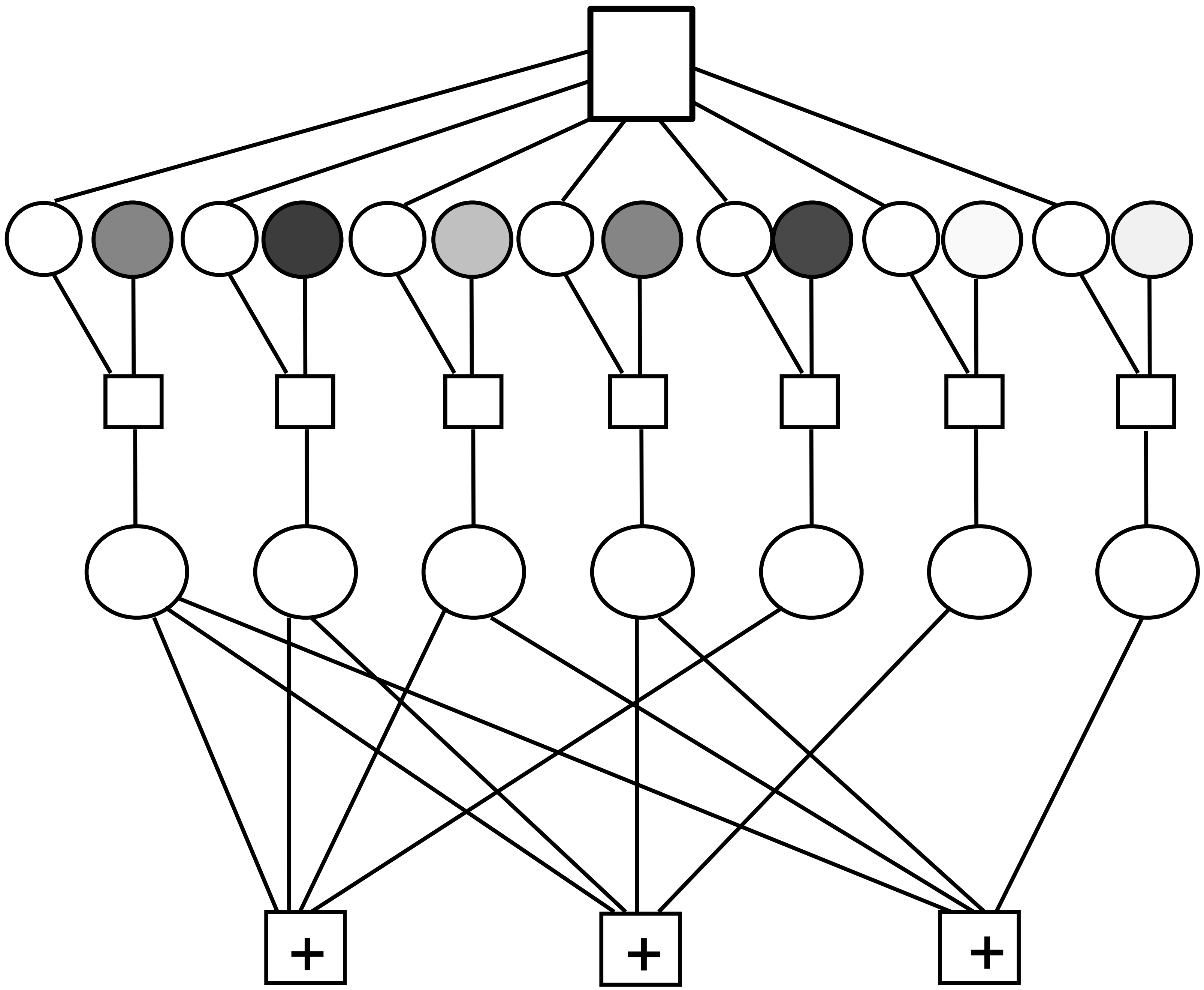
x_2	x_4	C_b
0	0	1.2
0	1	1.7
0	2	3.2
1	0	1.9
1	1	0.6
1	2	1.4

Connect "Cost Variables" to a total cost constraint

Outline

- Review of factor graphs for optimization and inference, and the min-sum Belief Propagation (BP) algorithm
- Gravel and Elser's "Divide & Concur" algorithm interpreted as a message-passing algorithm
- Decoders for Low-Density Parity Check (LDPC) Codes
  - Divide & Concur Decoder
  - "Difference-Map Belief Propagation" (DMBP) Decoder
- Simulation Results
  - *DMBP Decoder significantly improves error-floor performance compared with standard BP decoders, with similar complexity!*

Divide & Concur Decoder (Using Cost Variables)



Energy Constraint

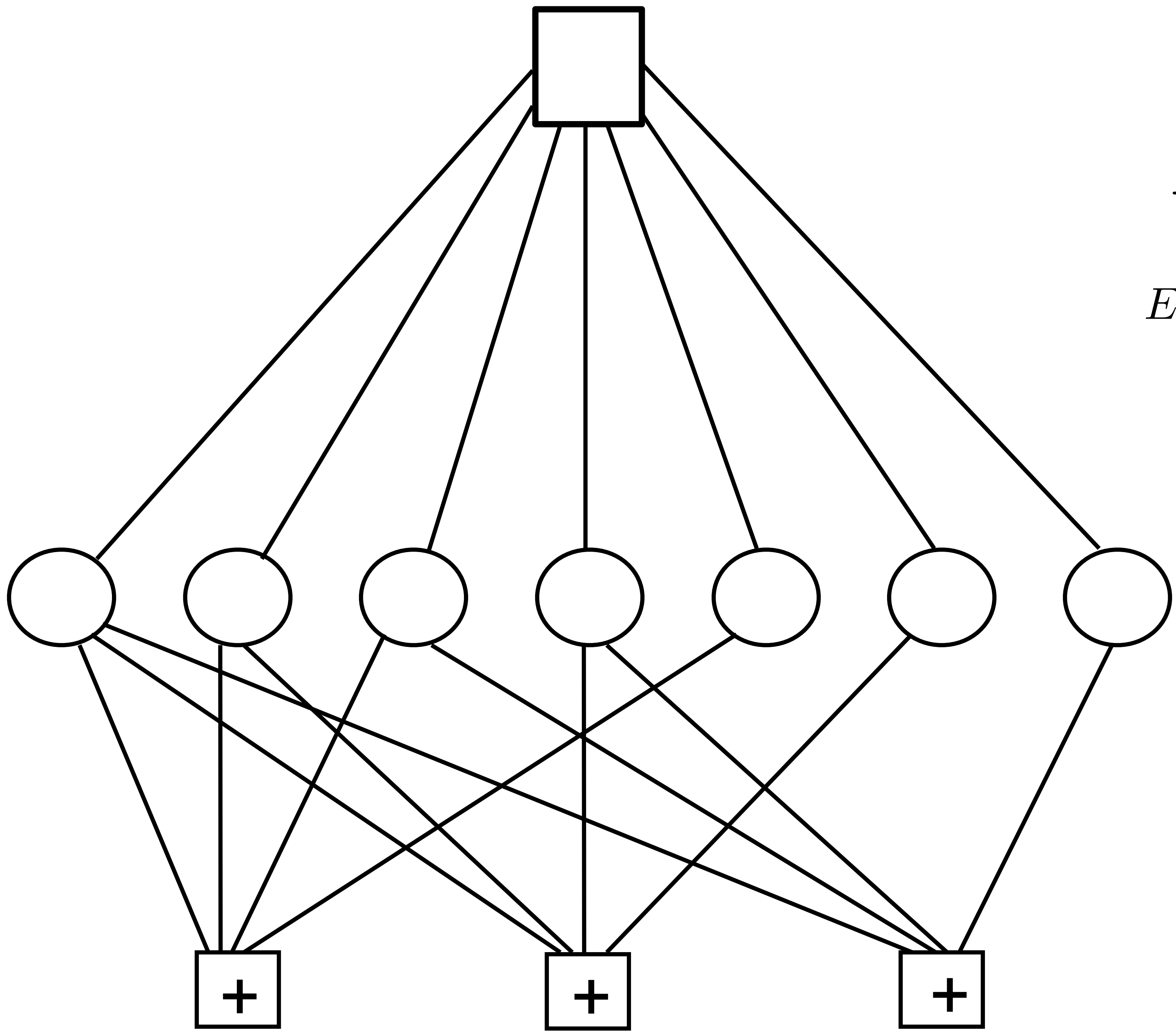
Costs / Observed Symbols

Transmitted Codeword Bits

Parity Check

Divide & Concur Decoder (Simplified Version)

(See also Gravel Ph.D. thesis 2009)



Energy Constraint

$$-\sum_{i=1}^{N} x_i L_i \le E_{\max}$$

$E_{\max} = -(1+\epsilon)\sum_i |L_i|$, with $0 < \epsilon \ll 1$

Energy constraint never satisfied, but terminate when you find a codeword

Transmitted Codeword Symbols $x_i = \pm 1$

Parity Checks

Difference-Map Belief Propagation

- D&C decoder performs OK, but not really better than sum-product BP.
- D&C decoder often decodes to incorrect codewords, something BP almost never does.
- But perhaps the "traps" that the difference-map avoids are related to the "trapping sets" that cause poor error-floor performance of BP decoders.
- Can we import the "difference-map" idea into a BP decoder?

Min-Sum BP

$$m_{a\to i}(t) = \left(\min_{j\in\mathcal{N}(a)\setminus i} |m_{j\to a}(t)|\right) \prod_{j\in\mathcal{N}(a)\setminus i} \operatorname{sgn}(m_{j\to a}(t))$$

$$b_i(x_i) = \sum_{a\in N(i)} m_{a\to i}(x_i)$$

$$m_{i\to a}(x_i) = b_i(x_i) - m_{a\to i}(x_i)$$

Divide & Concur

$$\boldsymbol{m}_{a\to}(t) = \boldsymbol{m}_{\to a}(t) + 2[P_D^a(\boldsymbol{m}_{\to a}(t)) - \boldsymbol{m}_{\to a}(t)]$$

$$b_i(x_i) = \frac{1}{|N(i)|}\sum_{a\in N(i)} m_{a\to i}(x_i)$$

$$m_{i\to a}(t+1) = b_i(t) - 1/2\,[m_{a\to i}(t) - m_{i\to a}(t)]$$

Difference-Map Belief Propagation

$$\boxed{m_{a\to i}(t) = \left(\min_{j\in\mathcal{N}(a)\setminus i} |m_{j\to a}(t)|\right) \prod_{j\in\mathcal{N}(a)\setminus i} \mathrm{sgn}(m_{j\to a}(t)).}$$

$$\boldsymbol{m}_{a\to}(t) = \boldsymbol{m}_{\to a}(t) + 2[P_D^a(\boldsymbol{m}_{\to a}(t)) - \boldsymbol{m}_{\to a}(t)]$$

$$b_i(x_i) = \sum_{a\in N(i)} m_{a\to i}(x_i)$$

$$\boxed{b_i(x_i) = Z \sum_{a\in N(i)} m_{a\to i}(x_i)}$$

$$b_i(x_i) = \frac{1}{|N(i)|} \sum_{a\in N(i)} m_{a\to i}(x_i)$$

$$m_{i\to a}(x_i) = b_i(x_i) - m_{a\to i}(x_i)$$

$$\boxed{m_{i\to a}(t+1) = b_i(t) - 1/2\,[m_{a\to i}(t) - m_{i\to a}(t)]}$$

Comments and Justifications

- Min-sum rule already overshoots in some sense
  - If there are three one's and a zero attached to a check, every bit will flip
- Wasn't clear whether BP's "belief is a sum" or D&C's "belief is an average" rule made more sense, so we compromise.
- Use D&C overshoot-correction rule.
- We also tried a sum-product version of DMBP, but it actually performed worse than the min-sum version!
  - This is surprising, because sum-product BP usually performs better than min-sum BP, and min-sum BP would otherwise be preferred because it is simpler to implement.

Outline

- Review of factor graphs for optimization and inference, and the min-sum Belief Propagation (BP) algorithm
- Gravel and Elser's "Divide & Concur" algorithm interpreted as a message-passing algorithm
- Decoders for Low-Density Parity Check (LDPC) Codes
  - Divide & Concur Decoder
  - "Difference-Map Belief Propagation" (DMBP) Decoder
- Simulation Results
  - *DMBP Decoder significantly improves error-floor performance compared with standard BP decoders, with similar complexity!*

Multi-stage Decoders

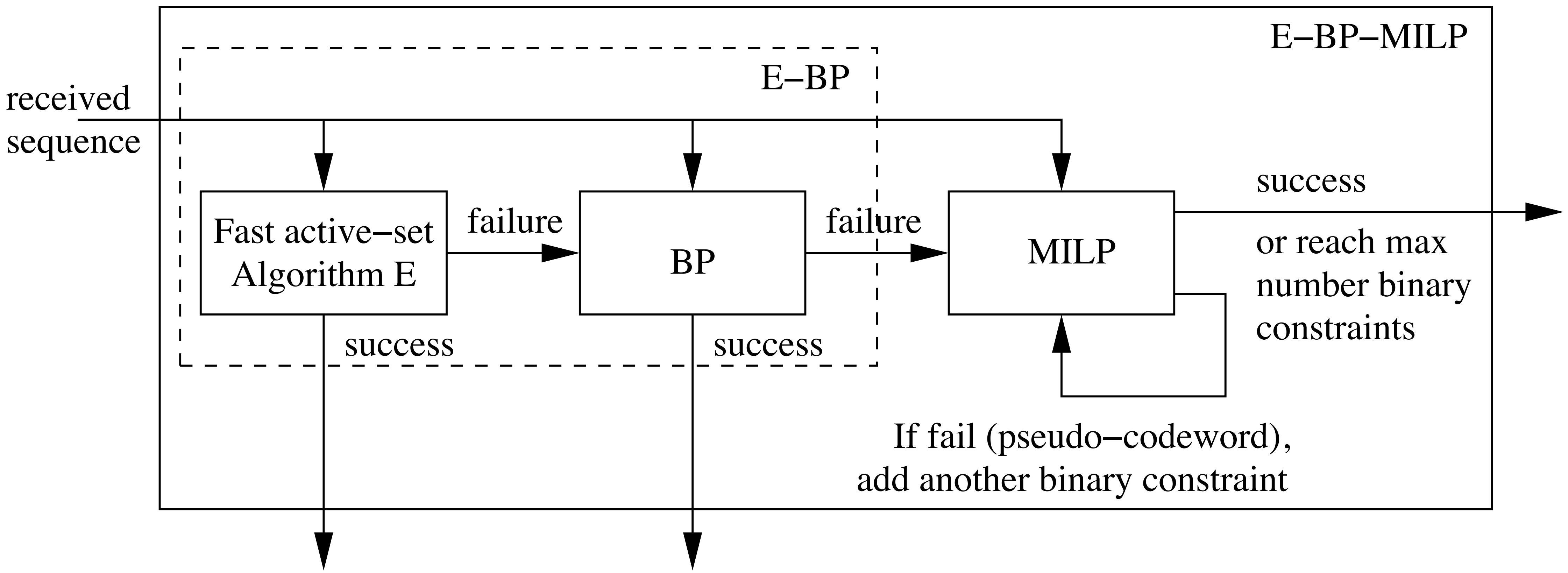
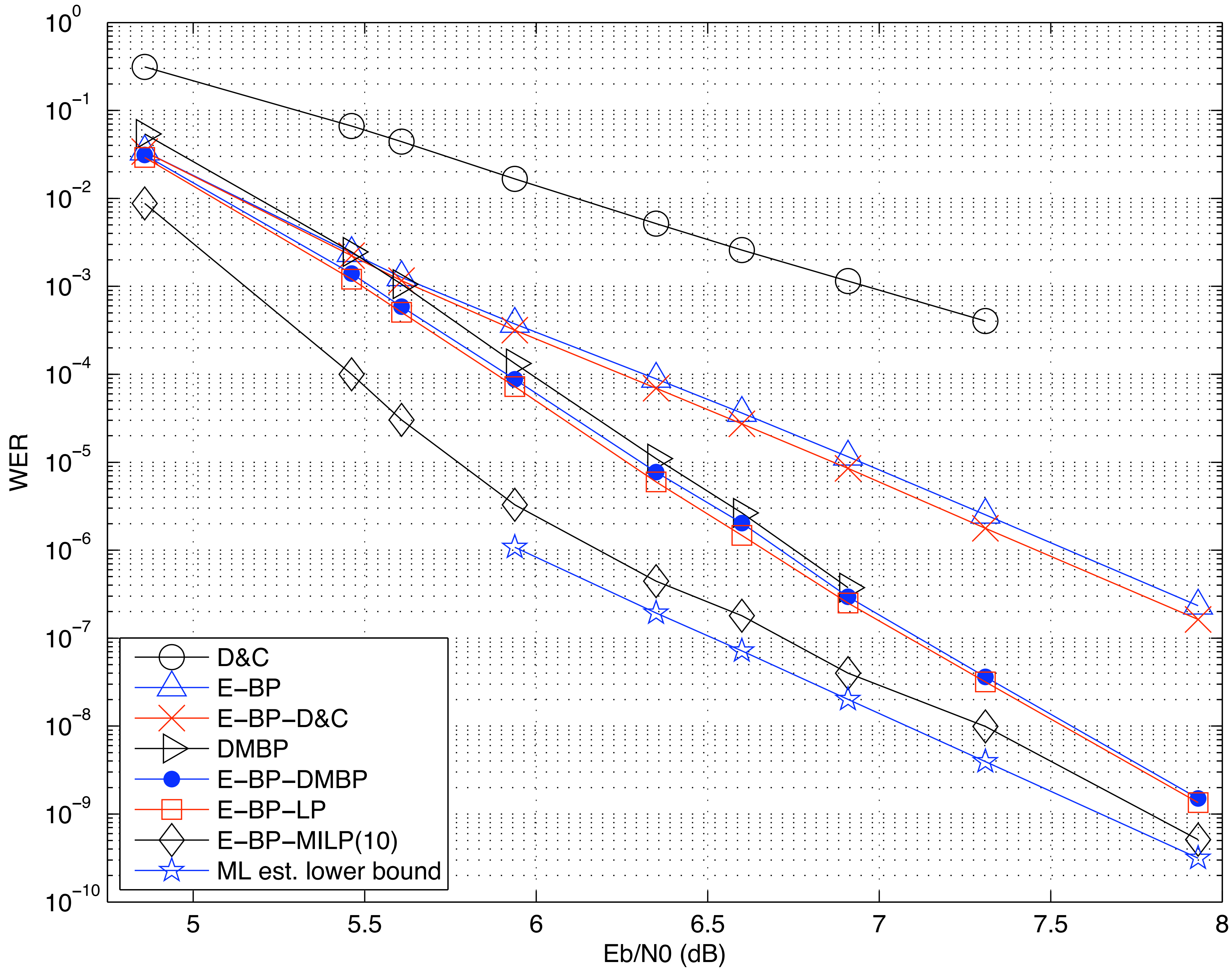


Fig. 1. Structure of an E-BP-MILP decoder.

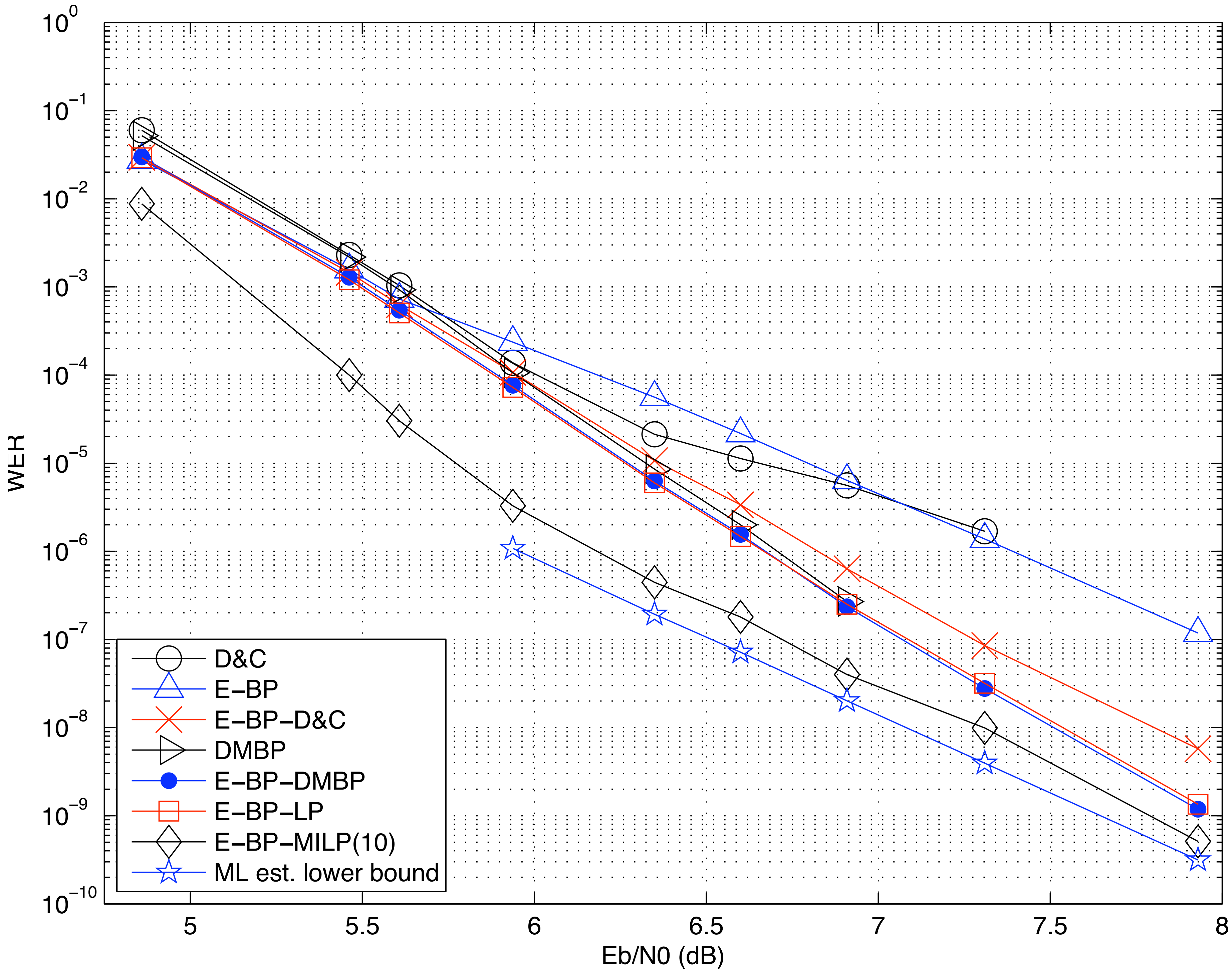
See Y. Wang, J.S. Yedidia, S.C. Draper, ISIT 2009

Length=1057, rate=0.77, random LDPC over BSC



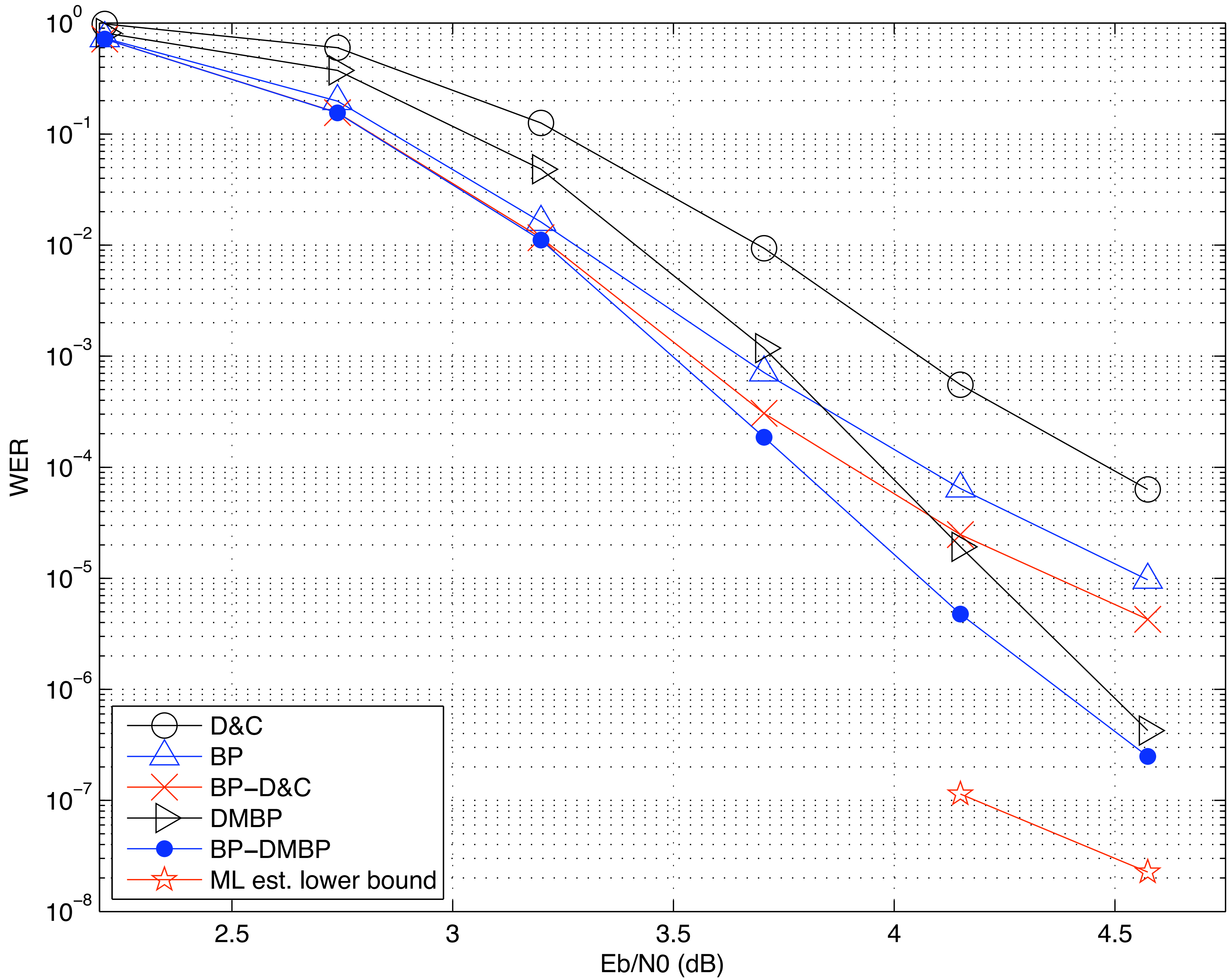
(a) Results when $T_{\max} = 50$ iterations

Length=1057, rate=0.77, random LDPC over BSC



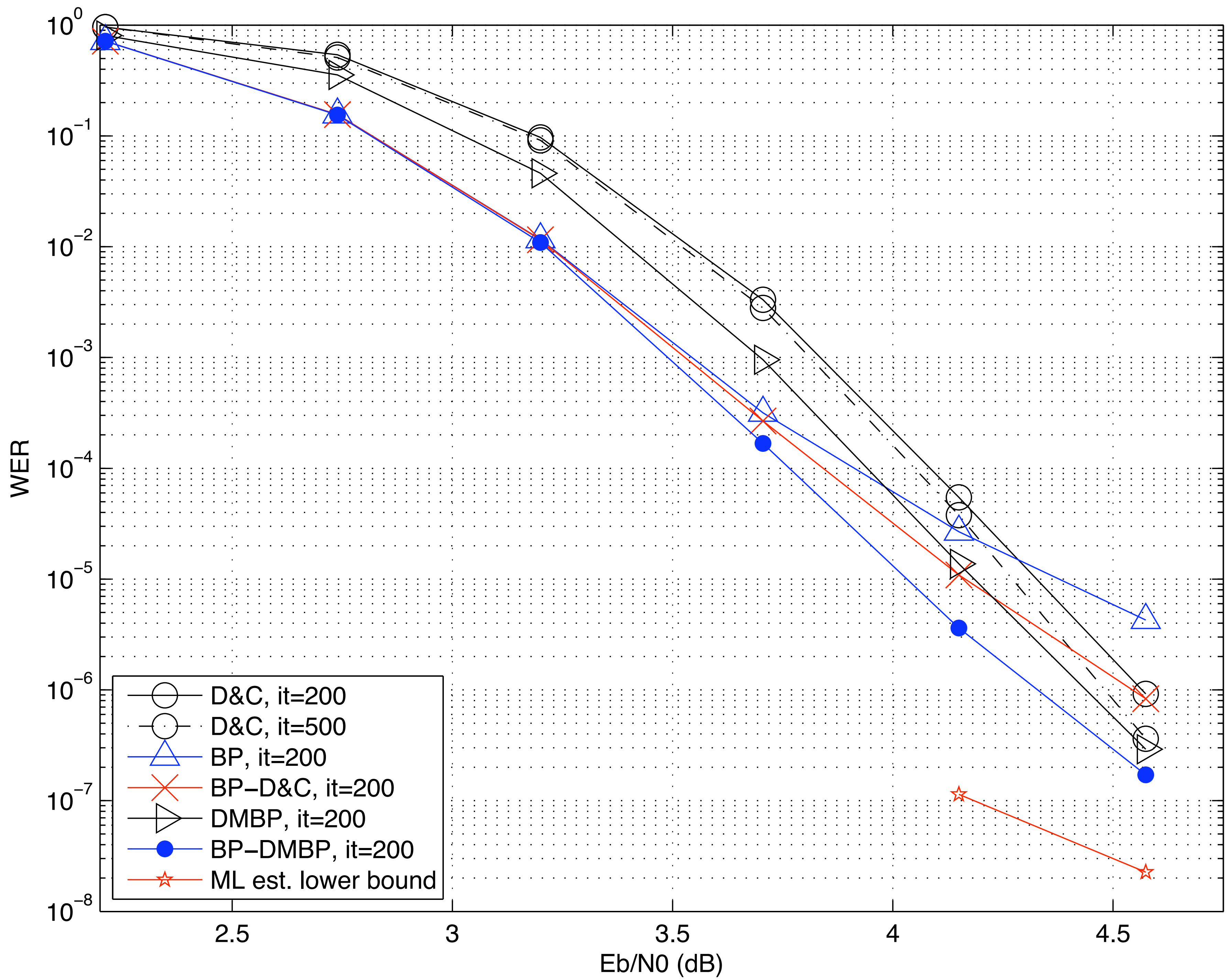
(b) Results when $T_{\max} = 300$ iterations

Length=1057, rate=0.77, random LDPC over AWGNC



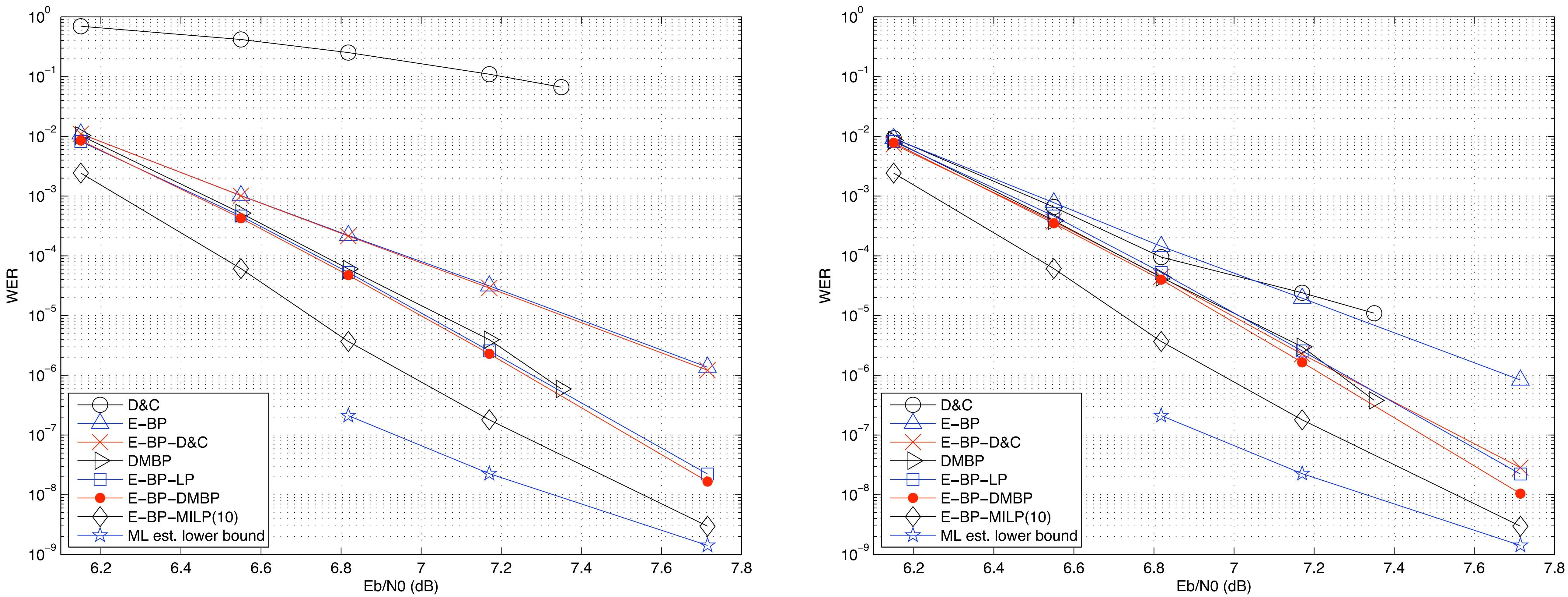
(a) Results when $T_{\max} = 50$ iterations

Length=1057, rate=0.77, random LDPC over AWGNC



(b) Results when $T_{\max} = 200$ or 500 iterations

Length=2209, rate=0.916, Array LDPC over BSC



(a) Results when $T_{\max} = 50$ iterations

(b) Results when $T_{\max} = 300$ iterations

Fig. 4. Error performance comparisons for a length-2209, rate-0.916 array LDPC code over the BSC.

Summary

- Gravel and Elser's Divide & Concur algorithm is an interesting competitor to Belief Propagation, that can handle a very wide variety of problems, including problems with continuous variables and with no local evidence. The difference-map dynamics of D&C lets it avoid local "traps."

- Divide & Concur can be usefully re-formulated as a message-passing algorithm.

- Divide & Concur decoders of LDPC codes are not very impressive, but simulations show that importing the difference-map idea into a min-sum BP decoder results in a significantly improved decoder compared to the standard sum-product BP decoder, with similar complexity.